Artificial Intelligence

LAB PROJECT SUBMISSION

Submitted to: Ms. Jhilik Bhattacharya

Submitted by:

Vanshika Babbar(102103297),Saksham Singla(102103298),

Sheral Singla(102103300),Manu Agarwal(102103318)

Problem Statement:

Creating a GUI Interface for CHATBOT using python and Natural Language Processing

---

Problem Description:

To design and develop an NLP-based chatbot that can effectively understand and respond to natural language input from users, providing relevant and accurate information or assistance while maintaining a conversational tone. The chatbot should be able to handle a variety of tasks such as answering FAQs, providing product or service information, making reservations, scheduling appointments, and offering personalized recommendations based on user preferences. The chatbot should also be able to handle multiple users concurrently and maintain context across different interactions with the same user. The primary goal is to improve customer satisfaction, reduce response times, and automate routine tasks.

NOTE:-

Our Chatbot primarily focuses on general **student – teacher interaction**

Code and Explanation:

```
Following is the approach :--
Theory + NLP concepts (stemming, tokenization, bag of words)
Create training data
PyTorch model and training
Save/load model and implement the chat
Next creating a GUI interface with the help of python tkinter
```

All the given steps are to be followed one by one

- **STEP-1** -Creating a **nltk_utils.py** file for all the NLP pre-processing work to be done

CHATBOT-MASTER > nltk_utils.py > ...

```python
import numpy as np
import nltk
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()

def tokenize(sentence):
    return nltk.word_tokenize(sentence)

def stem(word):
    return stemmer.stem(word.lower())

def bag_of_words(tokenized_sentence, words):
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1
    return bag
```

TOKENIZATION -- split sentence into array of words/tokens a token can be a word or punctuation character or a number

STEMMING -- finding the root form of the word

Example-
Words=[“organize” , ”organizes” , ”organizing”]

-->Words=[“organ” , ”organ” , ”organ”]

BAG_OF_WORDS -- return Bag of words array:

1-For each known word that exists in the sentence,0 otherwise

Example—

Sentence = [“hello” , ”how” , “are” , “you”]

Words = [“hi” , “Hello” , “I” , “you”, “bye” , “thank” , “cool”]

Bog = [0 , 1 , 0 , 1 , 0 , 0 , 0]

- **STEP-2** - Creating a **train.py** file for Training our model

```
with open('E:\pytorch-chatbot-master\CHATBOT-MASTER\intents.json', 'r') as f:
    intents = json.load(f)

all_words = []
tags = []
xy = []
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['tag']
    # add to tag list
    tags.append(tag)
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = tokenize(pattern)
        # add to our words list
        all_words.extend(w)
        # add to xy pair
        xy.append((w, tag))
```

Here firstly ,our input data for model train is read in form of json file(intents.json)

1. Now looping through our intents file and separating the tags from data ,

2. Iterating through the patterns to append the [all_words] list with tokenize sentences and [xy] list with [(tokenize_sentence),(tags)]

```
# stem and lower each word
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))
```

3. Ignoring all the punctuations from the sentences (given as input) and removing all the duplicates from the words and tags.

```
# create training data
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    # y: PyTorch CrossEntropyLoss needs only class labels, not one-hot
    label = tags.index(tag)
    y_train.append(label)

X_train = np.array(X_train)
y_train = np.array(y_train)

# Hyper-parameters
num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
print(input_size, output_size)
```

4. Splitting our data is done in [X_train , Y_train] where X_train is sentences and Y_train is basically our labels

5. Hyper-parameters are also set for our model

```
class ChatDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train

    # support indexing such that dataset[i] can be used to get i-th sample
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    # we can call len(dataset) to return the size
    def __len__(self):
        return self.n_samples

dataset = ChatDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=0)
```

6. Here a class is created for our custom dataloader and train_loader is set where class as dataset is given input and all other parameters are also given as input.

```
model = NeuralNet(input_size, hidden_size, output_size).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        # Forward pass
        outputs = model(words)
        # if y would be one-hot, we must apply
        # labels = torch.max(labels, 1)[1]
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 100 == 0:
        print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')


print(f'final loss: {loss.item():.4f}')
```

7. Our model is imported from different file and set as model=NeuralNet and optimizer & CrossEntropyloss is also set.

8. A training loop is created with num_epochs are defined and loss are printed for every epoch whereas final loss is also printed.

```
data = {
"model_state": model.state_dict(),
"input_size": input_size,
"hidden_size": hidden_size,
"output_size": output_size,
"all_words": all_words,
"tags": tags
}

FILE = "data.pth"
torch.save(data, FILE)

print(f'training complete. file saved to {FILE}')
```

9. here our model is saved in different file named as data.pth

- **STEP-3** - Creating a **model.py** file for model creation

CHATBOT-MASTER > model.py > ...

```
import torch
import torch.nn as nn


class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        # no activation and no softmax at the end
        return out
```

10. Here a NeuralNet is created containing three linear layers and and relu activation function is applied after every layer whereas no softmax is applied at end

- **STEP-4** - Creating a **chat.py** file for taking input and from user end doing all the process

```
bot_name = "ROZITA"
def get_response(msg):
    sentence = tokenize(msg)
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)

    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    if prob.item() > 0.75:
        for intent in intents['intents']:
            if tag == intent["tag"]:
                return random.choice(intent['responses'])

    return " I do not understand..."
```

Here our input sentence from user in taken as input in above function and then all further steps are followed:-

1. Tokenization takes place
2. Bag of words and then our size is reshaped because this is our input shape in model
3. Converting them into tensors and giving it to our model
4. Now taking output as probability then applying softmax to it
5. If prob>0.75 then it takes from intent file and then matching its tag from it then giving response from the responses variable in intent.json
6. Else prob<0.75 then Chatbot gives output as “I do not understand…”

- **STEP-5** - Creating a **app.py** file for making the GUI by applying tkinter library from python

```
class ChatApplication:
    def __init__(self):
        self.window = Tk()
        self._setup_main_window()

    def run(self):
        self.window.mainloop()

    def _setup_main_window (self):
        self.window.title("Chat")
        self.window.resizable (width=False, height=False)
        self.window.configure (width=470, height=550, bg=BG_COLOR)
# head label
        head_label = Label(self.window, bg=BG_COLOR, fg=TEXT_COLOR,
        text="Welcome", font=FONT_BOLD, pady=10)
        head_label.place(relwidth=1)
#tiny divider
        line = Label(self.window, width=450, bg=BG_GRAY)
        line.place(relwidth=1, rely=0.07, relheight=0.012)
        self.text_widget = Text (self.window, width=20, height=2, bg=BG_COLOR, fg=TEXT_COLOR,
                                 font=FONT, padx=5, pady=5)
        self.text_widget.place(relheight=0.745, relwidth=1, rely=0.08)
        self.text_widget.configure(cursor="arrow", state=DISABLED)

        scrollbar = Scrollbar(self.text_widget)
        scrollbar.place(relheight=1, relx=0.974)
        scrollbar.configure (command=self.text_widget.yview)
```

```
# bottom label
bottom_label = Label(self.window, bg=BG_GRAY, height=80)
bottom_label.place(relwidth=1,rely=0.825)

self.msg_entry = Entry(bottom_label, bg="#2C3E50", fg=TEXT_COLOR, font=FONT)
self.msg_entry.place(relwidth=0.74, relheight=0.06, rely=0.008, relx=0.011)
self.msg_entry.focus()
self.msg_entry.bind("<Return>", self._on_enter_pressed)
```

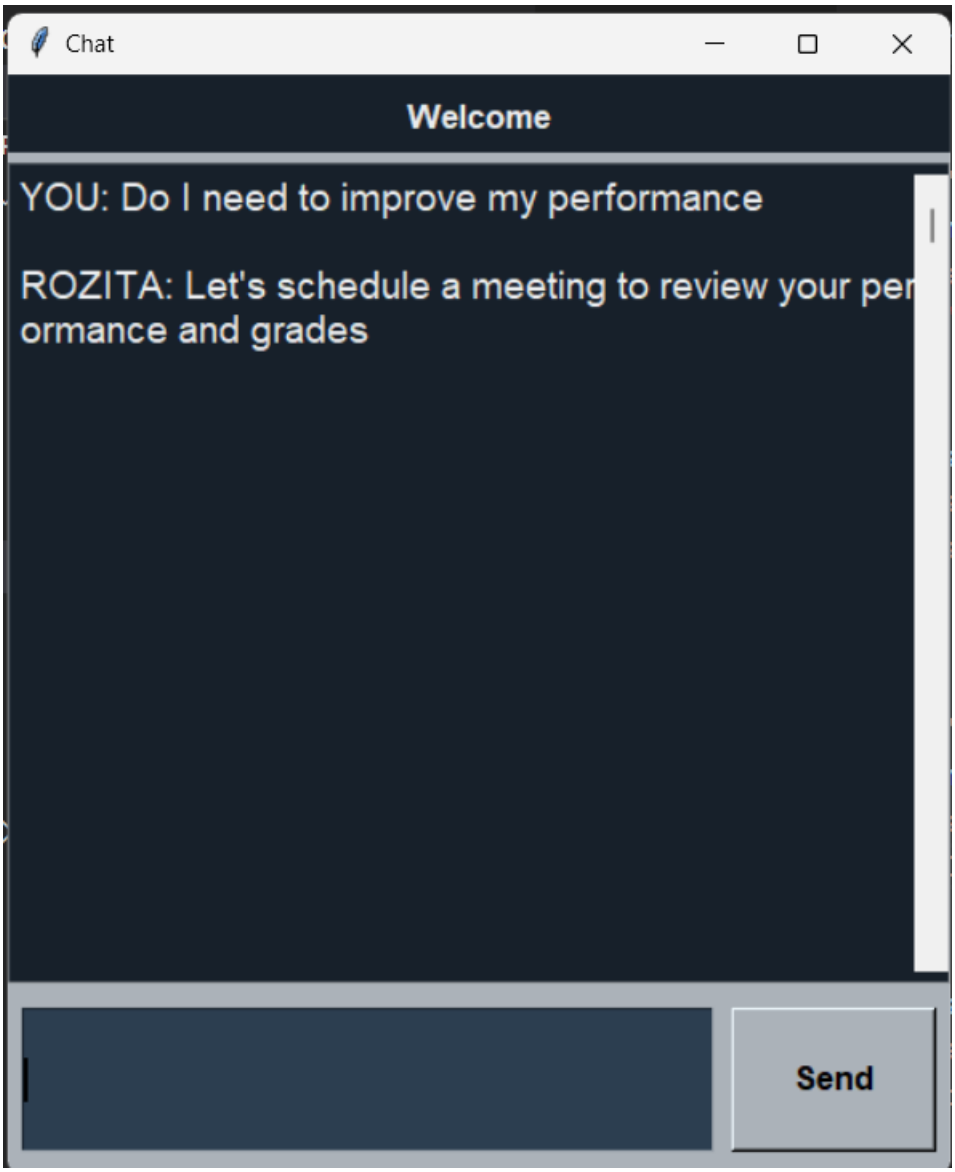
```
    send_button = Button (bottom_label, text="Send", font=FONT_BOLD, width=20, bg=BG_GRAY,
                          command=lambda: self._on_enter_pressed (None))
    send_button.place(relx=0.77, rely=0.008, relheight=0.06, relwidth=0.22)

def _on_enter_pressed(self, event):
    msg = self.msg_entry.get()
    self._insert_message(msg, "YOU")
def _insert_message(self, msg, sender):
    if not msg:
        return
    self.msg_entry.delete(0, END)
    msg1 = f"{sender}: {msg}\n\n"
    self.text_widget.configure (state=NORMAL)
    self.text_widget.insert (END, msg1)
    self.text_widget.configure(state=DISABLED)

    msg2 = f"{bot_name}: {get_response(msg)}\n\n"
    self.text_widget.configure (state=NORMAL)
    self.text_widget.insert (END, msg2)
    self.text_widget.configure(state=DISABLED)

    self.text_widget.see(END)
```

```
if __name__ == "__main__":
    app = ChatApplication ()
    app.run()
```

The given code is a Python implementation of a chat application using the tkinter library for the graphical user interface. It defines a ChatApplication class with methods for setting up the main window, handling user input, and displaying messages. The application allows the user to enter text messages and displays both the user's messages and the bot's responses in a text widget. When the user presses the send button or hits the enter key, the application sends the user's message to the bot and displays the conversation history in the text widget.

Conclusion:

Through NLP, the chatbot can understand and respond to human language in a more human-like manner, making interactions more seamless and intuitive. Deep learning algorithms enable the chatbot to continually learn and improve its responses based on user interactions, leading to more accurate and personalized conversations.


Chat
Welcome
YOU: Do I need to improve my performance
ROZITA: Let's schedule a meeting to review your per
ormance and grades
Send